

Contents

	Introduction	iv
Section 1	Computer Systems	
1	Data representation	1
2	Data transmission	15
3	Hardware	25
4	Software	49
5	The internet and its uses	62
6	Automated and emerging technologies	80
Section 2	Algorithms, programming and logic	
7	Algorithm design and problem solving	98
8	Programming	113
9	Databases	133
10	Boolean logic	137

	REVISED	TESTED	EXAM READY
1	☐	☐	☐
15	☐	☐	☐
25	☐	☐	☐
49	☐	☐	☐
62	☐	☐	☐
80	☐	☐	☐
98	☐	☐	☐
113	☐	☐	☐
133	☐	☐	☐
137	☐	☐	☐

Introduction

Welcome to the *Cambridge IGCSE™ and O Level Computer Science Study and Revision Guide Second Edition*. This book has been written to help you revise everything you need to know for your Computer Science examinations, alongside the *Cambridge IGCSE and O Level Computer Science Second Edition Student's Book*. Following the new Computer Science syllabus (first exams in June 2023), this book covers all the key content along with sample questions and answers and exam-style practice questions. Sample practice papers also appear at the end of the book and online at <https://www.hoddereducation.co.uk/cambridgeextras>. These practice papers, written by the authors, are slightly longer than the actual Cambridge IGCSE papers to ensure questions covering more of the syllabus can be offered. There is also an online glossary for key words that appear in red bold in this book.

How to use this book

2

Data transmission

Key objectives

The objectives of this chapter are to revise:

- methods of data transmission:
 - data packets
 - structure of data packets
 - packet switching
 - serial, parallel, simplex, half-duplex and full-duplex
 - universal serial bus (USB)
- error detection to include parity checks, checksum, echo checks and Automatic Repeat Requests (ARQ)
- encryption to include symmetric and asymmetric encryption

Key objectives

A summary of the main information.

2.1 Types and methods of data transmission

REVISED

2.1.1 Data packets

Data is broken up into **packets** before it is transmitted. Data packets are split up into:

- **packet header** (containing the IP address of the sending station and receiving station, the sequence number of the packet, so it can be reassembled, and packet size, to ensure the receiving station knows that the whole packet has been received)
- **payload** (the actual data)
- **packet trailer** (containing **cyclic redundancy check (CRC)** (error check) and a way of identifying the end of the data packet).

CRC is found by adding 1-bits and send as a hex value

Data packets allow data to be sent in manageable chunks that can be sent along the most efficient route from A to B. **Routers** (known as **nodes**) are used to control the path a data packet takes from sending station to receiving station. This is called **packet switching** where each data packet can take a different route; each route taken is independent of each other. Since data packets take different routes, they could arrive at their destination in the wrong order. A sequence number in the packet header allows all the data packets to be reassembled in the right order at the receiving station. **Hop numbers** are added to packet headers to ensure they don't 'bounce around' from router to router and eventually become effectively lost.

Benefits of packet switching	Drawbacks of packet switching
<ul style="list-style-type: none">• There is no need to tie up a single communication line.• It is possible to overcome failed, busy or faulty lines by simply re-routing packets.• It is relatively easy to expand package usage.• A high data transmission rate is possible.	<ul style="list-style-type: none">• Packets can be lost and need to be re-sent.• The method doesn't work well with real-time streaming (for example, a live sporting event being transmitted over the internet).• There is a delay at the destination whilst the packets are being re-ordered.

It is illegal to photocopy this page

It is illegal to photocopy this page

Sample question and answer

REVISED

Exam-style questions with sample student answers to show how the question can be answered. In the Theory section, high-level answers (in blue) are strong answers while low-level (in purple) answers require more revision.

Sample questions and answers

A video conference is taking place between delegates in the USA and Mauritius. Packet switching is used to send video and sound data between delegates.

Describe:

- the potential problems with sound and video quality
- how the problems in part a) could be overcome.

[6]

Sample high-level answer

- Video conferencing is in real time. Video and audio data is split up into data packets before it is sent over the internet. This means that each data packet could potentially follow a different route from delegate to delegate. Data packets could therefore take different times from sender to recipient, or some may even become 'lost'. The arriving packets also need to be reassembled into the correct sequence, which can also cause a time delay. All of this could lead to gaps in data received, freezing of images, drop out of voice and mouth 'out of synchronisation' – this would be caused by audio and video data taking different times from sender to receiver.
- Since the problems seem to be due to timing issues, the best solution would be to eliminate these time differences. A single path for all data packets would resolve this; that could be achieved using a dedicated communication link between delegates.

A mention of circuit switching could also be accepted here if a candidate has gone outside the syllabus.

Sample low-level answer

- Sometimes you get echo or lag on the sound or it may even drop out on occasions. The video and sound don't always synchronise where a person's lips don't match with the words heard. Sometimes the image just freezes.
- This can be overcome using better cameras and microphones and getting a faster internet connection.

Exam-style questions

- Explain how packet switching could be used to download some software from a website. [4]
- Data being shared between two computers is split up into a number of data packets prior to transmission. The data packets contain a header and a trailer.
 - Give **three** pieces of information found in the header. [3]
 - The trailer contains a cyclic redundancy check (CRC).
 - Name another item found in the trailer. [1]
 - The following data is being sent in the payload:
11110000 00111111 00110011 11001100 11111110 11101110
00001100 11011111 11001001
Use this data to show how CRC is used to ensure no errors occurred during data transmission. [3]
 - Explain how it is possible for data packets to be lost during their transmission across a network. [2]
 - Describe how it is possible to deal with lost packets so that they don't cause network problems, such as 'clogging up' the system. [2]

Tips

Since this is a 'describe' question, it is necessary to give all the main facts, features and/or characteristics about video conferencing when using packet switching. It is important to notice the reference to part a) in part b) – this means that your answer to part b) must refer back to potential problems identified in part a).

Teacher's comments

The first answer shows that the student has realised the connection between video conferencing and packet switching. They have correctly referred to 'lost' packets or packets out of order as a reason why there can be sound and vision issues. Their solution of using circuit switching (a dedicated route) is the kind of answer you would expect from a top-level answer (it goes slightly beyond the syllabus, but would still gain credit).

The second answer hasn't made the link between video conferencing and packet switching. But they have mentioned some of the problems that can occur without mentioning how or why they occur. The answer would gain two of the available marks in part a). One mark could also be awarded in part b) for suggesting faster internet could go some way to resolving the issue.

Tips

Advice on how to answer the question.

Teacher's comments

Feedback from a teacher showing what was good and what could be improved in the student answers.

It is illegal to photocopy this page

16

Cambridge IGCSE™ and O Level Computer Science Study and Revision Guide Second Edition

Exam-style questions

Exam-style questions for you to try and see what you have learned.

Answers

Outline answers to the exam-style questions and Practice Papers 1A and 2A from page 163. Practice Paper 1B and 2B and their answers are online.

It is illegal to photocopy this page

Study tips

REVISED

Know what questions you may be asked

Find a copy of the IGCSE Computer Science syllabus that you have been studying and will take the exam for and use your revision guide and textbook. The list of topics will make an excellent checklist for your revision. If you find something that you do not understand or have not made any notes about, stop and find out about it while you remember. Anything that is mentioned in the syllabus could be used in a question.

Read and understand questions

What are you being asked to do?

- 1 Read the question thoroughly.
- 2 Understand the type of instruction you are being given: *complete*, *describe*, *explain*, *give*, *state* and *suggest* all require different actions (see later for more details about key instructions).
- 3 If the question makes use of a specific scenario, make sure all your answers are relevant to the scenario. For example, if the question is about website security measures taken by a bank, suggesting taking the website offline outside banking hours would not be an appropriate response.
- 4 Decide on the information required but remember that most answers will require more than just a single word. If you have finished well before the time allotted, you may not have answered your questions in enough detail. Go back and check your answers.
- 5 Always use the correct technical terms and avoid the use of trade names at all costs (no marks will be awarded for answers such as *Google*, *Word* or *Norton* for example). If you are referring to using Word, your answer should discuss how to use a word processor with no mention of any trade names.
- 6 Decide how much information is required by checking out the instructions given and also by looking at the number of marks allocated to the question (for example, if a question is awarded 5 marks, then your answer will need to include at least five marking points).

Know your subject

- Make sure that you understand computer science terms and that you can spell them correctly: for example, *phishing*, *pharming*, *verification*, *autonomous* and so on.
- Learn the definitions and be able to provide examples to show your understanding. There are certain items that need learning so that you can apply them, such as validation checks, types of translator and so on.
- Do not ignore items in the syllabus that you do not understand or do not have notes on. Do some research to fill all your gaps in knowledge.
- Beware of cramming facts into your brain without understanding them or being able to apply them. You need to revise but try and make it interesting rather than just reading and re-reading your notes. Remember, writing things down, rather than just reading about them, means you are more likely to recollect important facts. Always practice exam-style questions (including looking at past papers) and remember to time yourself since you do not want to run out of time when you do the actual exam.

- Do not try to learn too much at the last minute. Use the best revision technique that suits you. It is often a good idea to write out revision cards so you can do a last-minute check. For example, a binary arithmetic check card could look something like this:

Binary arithmetic

- Binary headings (128, 64, 32, 16, 8, 4, 2, 1)
- Remember $1 + 1 = 0$ carry 1; $1 + 1 + 1 = 1$ carry 1, etc.
- Two's complement uses -128 in left-most bit position
- Shifting binary numbers left is the same as multiplying by factors of 2
- Shifting binary numbers right is the same as dividing by factors of 2

- Questions asked will be within the syllabus boundaries. If something appears new to you, then the question will give you lots of guidance so you can call on facts that you should already know.
- Make sure that you read all of the question carefully.
- Make sure that you understand the work that you have done – if you only try to remember it, you will not be able to apply your knowledge.
- Mark allocations are there to help you decide on how to use your time sensibly, so do not spend 30 minutes on a question worth only two marks.

Help yourself

- Get a good night's sleep or take a decent lunch break before doing the exam. The most important thing that goes into the exam room is your brain and it does not function well if you have not had enough rest. Think of exam preparation in the same way as preparation for taking part in a sporting event.
- Stay calm. Everyone gets nervous in some way, so work out how best to relax yourself before starting the exam.
- Read the whole paper through first and do not write anything at this stage. This will help you gather your thoughts and get some idea of how much effort each question is going to need. Remember, you do not have to answer questions in numerical order. Answer those questions first that you are confident about. This will boost your confidence and will help when you come to the more challenging questions.
- Remember to read each question thoroughly before answering it. Highlight or underline key words in the question so you do not lose marks needlessly by missing out something important (for example, you may have to give an answer in GiB and you give your answer in KiB which could lose you marks – this could be avoided by highlighting the word 'GiB').
- Use clear English and good handwriting.
- Do not use correction fluid and cross out unwanted work neatly. If you have written the answer on a different page, remember to show very clearly where your answer can now be found.
- Once finished, if you have any time left, go back and check your answers. You may have missed something out which could make a difference.
- Attempt to answer all questions. Leaving blank spaces is not a good idea. You just might get credit by writing down something which you think could be relevant.

Command words

The use of certain words in questions gives you a strong clue about how to answer them. The following table gives a list of common key words used, their meaning and a sample question.

Key word	Meaning	Example
Calculate	Work out from given facts, figures or information.	Calculate the size of the file needed to store 1200 photos which are each 12 MiB in size.
Compare	Identify/comment on similarities and/or differences.	Compare the features of random access memory (RAM) and read-only memory (ROM).
Define	Give precise meaning.	Define what is meant by the term packet switching.
Demonstrate	Show how or give an example.	Demonstrate how hexadecimal is converted to binary.
Describe	State the points of a topic/give characteristics and main features.	Describe how piezoelectric technology is used by an inkjet printer to produce a hard copy of a photograph.
Evaluate	Judge or calculate the quality, importance, amount or value of something.	Evaluate the importance of using robotics in medicine.
Explain	Set out purposes or reasons; make the relationships between things evident; provide why and/or how and support with relevant evidence.	Explain why packet switching may not be a good way of transmitting real time data.
Give	Produce an answer from a given source or from memory.	Give a reason why hexadecimal notation is used.
Identify	Name/select/recognise.	Identify the type of memory used by a computer to store the start-up routines.
Outline	Set out the main points.	Outline the role of registers and buses in the fetch, decode, execute cycle.
Show	Provide structured evidence that leads to a given result.	Show how to convert a denary number to a hexadecimal number.
State	Express in clear terms.	State the advantages of using USB-C connectivity rather than using USB-A connectivity.
Suggest	Apply knowledge and understanding to situations where there are a range of valid responses in order to make proposals/put forward considerations.	Suggest how data entry for a name could be verified and validated.

Assessment

REVISED

The information in this section is taken from the Cambridge IGCSE and O Level Computer Science syllabuses (0478/0984/2210) for examination from 2023. You should always refer to the appropriate syllabus document for the year of examination to confirm the details and for more information. The syllabus document is available on the Cambridge International website at: www.cambridgeinternational.org

There are two examination papers:

	Paper 1 Computer systems	Paper 2 Algorithms, programming and logic
Duration	1 hour 45 minutes	1 hour 45 minutes
Marks	75 marks	75 marks
Percentage of overall marks	50%	50%
Syllabus topics examined	1–6	7–10

7

Algorithm design and problem solving

Key objectives

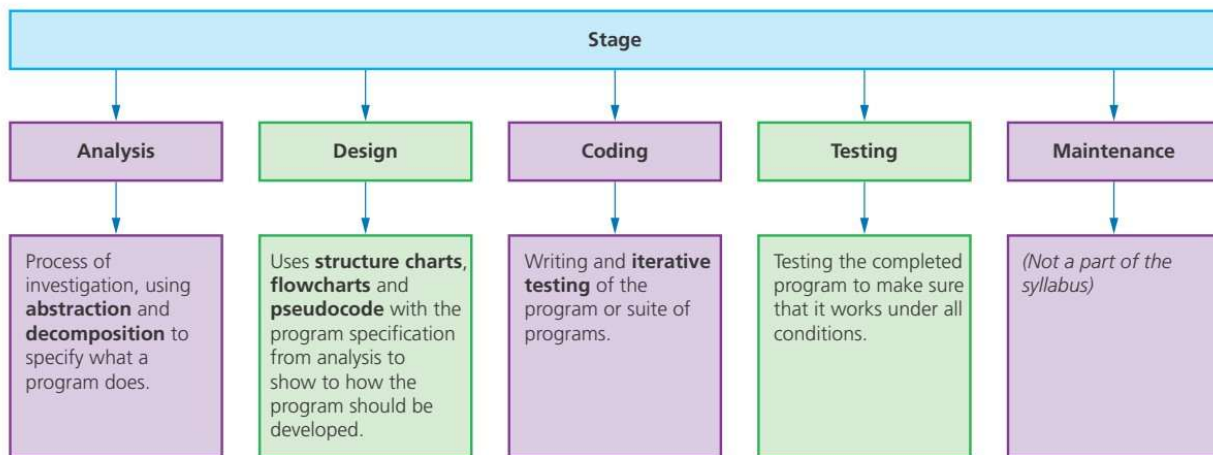
The objectives of this chapter are to revise:

- the stages in the program development cycle:
 - analysis, design, coding, testing
- computer systems and sub-systems
- problem decomposition into component parts
- methods used to design and construct solutions to problems
- the purpose of an algorithm and the processes involved in it
- standard methods of solution:
 - totalling, counting, finding maximum, minimum, average, linear search, bubble sort
- validation and verification checks when data is input
- use of different types of test data including:
 - documentation of a dry run using a trace table
- writing, amending identifying, and correcting errors in:
 - flowcharts, programs, pseudocode

7.1 Program development life cycle

REVISED

The program development life cycle is divided into five stages: **analysis**, **design**, **coding**, **testing** and **maintenance**.



Sample questions and answers

REVISED

The program development life cycle is divided into different stages. Analysis is the first stage.

- a) Describe the purpose of analysis.
- b) Identify and describe **two** other stages of the program life cycle. [6]

Sample high-level answer

- a) Analysis is used to investigate the problem that requires a solution and find out what the program needs to do to provide that solution.
- b) Two other stages are:
 design – document the tasks required for the solution using structure charts and pseudocode
 coding – write the programs that are required.

Sample low-level answer

- a) A systems analyst will use interviews and questionnaires to find out what is needed.
- b) The stages of program development are analysis, design, coding and testing. There is lots to do as the programs need to be written and fully tested.

Tips

Part a) of this question asks for the purpose, so your answer must state why the analysis stage is needed, not what actions are to be performed. It is always very important to read the question carefully. Part b) of this question involves 'identify'. This means your answer must either name the two stages of the program life cycle or clearly identify each one with a short statement.

Teacher's comments

The first answer would gain full marks since there is a clear description of the purpose of analysis. Two other stages of types of the program life cycle have been identified and described with each description clearly shown for the stage identified.

The second answer would gain one mark for part a) since the statement 'analysis is to find out what is needed' partially describes the purpose of analysis. Including the methods used answered a different question about how analysis is carried out. For part b) all the stages of the program life cycle are identified not just the two required. There is a description of the actions required but it is unclear which stage these apply to. A maximum of two marks could be awarded.

7.2 Computer systems, sub-systems and decomposition

REVISED

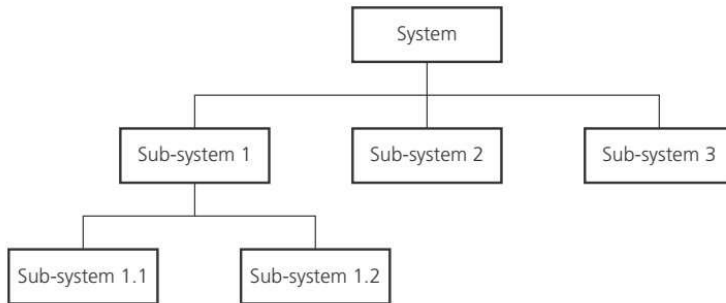
Each **computer system** is made up of software, data, hardware, communications and people. Each computer system can be divided up into a set of sub-systems and each sub-system can be further divided into sub-systems and so on until each sub-system just performs a single action. The process of **decomposition** into sub-systems so that a system can be more easily represented and understood is the basis of **top-down design**.

Any problem that uses a computer system for its solution needs to be decomposed into its component parts. These are **inputs**, **processes**, **outputs** and **storage**. The methods used to design and construct a solution are:

- structure diagrams
- flowcharts
- pseudocode.

7.2.1 Structure diagrams

A **structure diagram** shows hierarchically how each computer system can be divided up into a set of sub-systems.



7.2.2 Flowcharts

A **flowchart** shows diagrammatically the steps required to complete a task and the order that they are to be performed. These steps, together with the order, are called an **algorithm**.

Flowcharts are drawn using standard flowchart symbols as shown in this table.

Use	Symbol	Description
Terminator Start/Stop		Used at the beginning and end of each flowchart. At least two outputs.
Process		Used to show actions, for example, when values are assigned to variables.
Input/output		The same flowchart symbol is used to show the input of data and output of information.
Decision		Used to decide which action is to be taken next. These can be used for selection and repetition/iteration. There are always two outputs from a decision flowchart symbol.
Flow lines		Used to show the direction of flow.

7.2.3 Pseudocode

Pseudocode is a simple method of showing an algorithm. It describes what the algorithm does by using English key words that are very similar to those used in a high-level programming language and meaningful identifier names.

Mathematical operators

Operator	Action
+	Add
-	Subtract
*	Multiply
/	Divide
^	Raise to the power
()	Group

Pseudocode statements

Pseudocode statement	Examples
Assignment A value is assigned to an item/variable using the $\leftarrow$ operator.	$\text{Cost} \leftarrow 10$ $\text{SellingPrice} \leftarrow \text{Price} + \text{Tax}$
Conditional 1 A condition that can be true or false: IF ... THEN ... ENDIF or IF ... THEN ... ELSE ... ENDIF For an IF condition the THEN path is followed if the condition is true, and the ELSE path if it is false (an ELSE may not be required). The end of the statement is followed by ENDIF	IF Age < 18 THEN OUTPUT "Child" ELSE OUTPUT "Adult" ENDIF
Conditional 2 A choice between several different values: CASE OF ... OTHERWISE ... ENDCASE For a CASE statement, the value of the variable decides the path taken. Several variables are usually specified. OTHERWISE path is taken for all other values. The statement is ended by ENDCASE	CASE OF Grade "A" : OUTPUT "Excellent" "B" : OUTPUT "Good" "C" : OUTPUT "Average" OTHERWISE OUTPUT "Improve" ENDCASE
Iteration 1 FOR ... TO ... NEXT a variable is set up, with a start value and an end value, this variable is incremented in steps until the end value is reached and the iteration finishes.	FOR Counter $\leftarrow$ 1 to 10 OUTPUT "*" NEXT Counter
Iteration 2 REPEAT ... UNTIL is used when the number of repetitions/iterations is not known, and the actions are repeated UNTIL a given condition becomes true. The actions in this loop are always completed at least once.	Counter $\leftarrow$ 0 REPEAT OUTPUT "*" Counter $\leftarrow$ Counter + 1 UNTIL Counter $\geq$ 10
Iteration 3 WHILE ... DO ... ENDWHILE is used when the number of repetitions/iterations is not known, and the actions are only repeated WHILE a given condition is true. If the WHILE condition is untrue then the actions in this loop are never performed.	Counter $\leftarrow$ 0 WHILE Counter < 10 DO OUTPUT "*" Counter $\leftarrow$ Counter + 1 ENDWHILE
Input INPUT used for data entry.	INPUT Name INPUT StudentMark
Output OUTPUT or PRINT used to display information.	PRINT "Your name is", Name OUTPUT Name1, "Ali", Name3
Nesting 1 Nested IF makes use of two IF statements; the second IF statement is part of the first ELSE or THEN path	IF Age < 18 THEN OUTPUT "Child" ELSE IF Age > 65 THEN OUTPUT "Senior" ELSE OUTPUT "Adult" ENDIF ENDIF

Pseudocode statement	Examples
Nesting 2 Nested iteration makes use of two loops; the second loop is inside the first loop.	<pre> FOR Number ← 1 to 10 OUTPUT Number FOR Counter ← 1 to Number OUTPUT "*" NEXT Counter NEXT Number </pre>

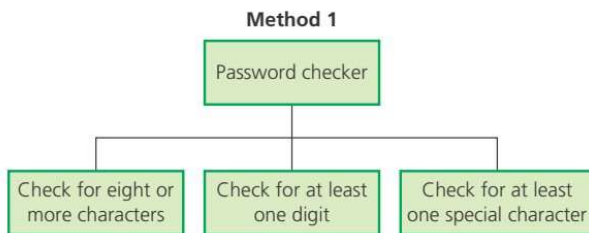
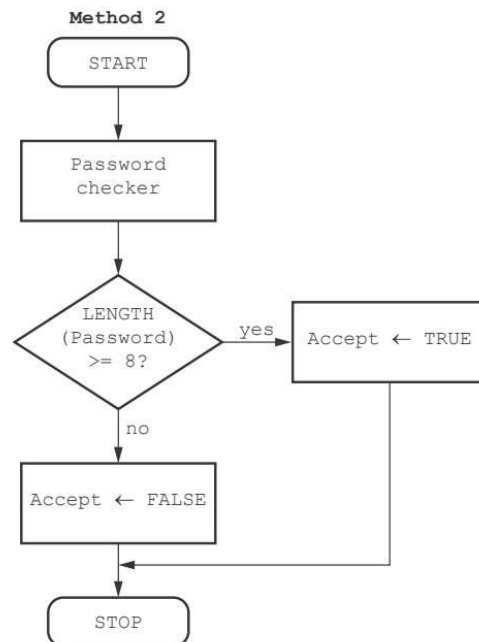
Comparison operators

Operator	Comparison
>	Greater than
<	Less than
=	Equal
>=	Great than or equal
<=	Less than or equal
◇	Not equal
AND	Both
OR	Either
NOT	Not

Sample questions and answers

REVISED

- a) A computer system can be decomposed into its component parts. One of the parts is input. Identify **two** other component parts.
- b) Formal methods are used to show a proposed solution.
- i) Identify the formal methods shown.



- ii) Identify another formal method that could be used to check if the password has more than eight characters.
- iii) Use the formal method you identified in part b)ii) to show the same routine to check if the password has more than eight characters.

[8]

Tip

Part a) of this question asks for two **other** component parts; you must not use the part mentioned in the question. It is surprising how many students do this and throw away a mark. Part b)i) and ii) of this question involves 'identify'. This means your answer must either name the methods or clearly identify each one with a short statement. Part b)iii) needs the same routine to be rewritten using pseudocode, not a programming language.

Sample high-level answer

- a) Process and output
- b) i) The two methods are:
 - Method 1 – structure diagram
 - Method 2 – flowchart
- ii) Pseudocode
- iii)

```
IF LENGTH (Password) >= 8
    THEN Accept ← TRUE
    ELSE Accept ← FALSE
ENDIF
```

Sample low-level answer

- a) input and output
- b) i) The two methods are:
 - Method 1 – structure chart
 - Method 2 – flow diagram
- ii) Code
- iii)

```
IF Pwd.length > 8:
    OK = Yes
ELSE: OK = No
```

Teacher's comments

The first answer would gain full marks since two other components are identified, the methods are identified correctly, and the routine rewritten exactly matched the flowchart given and used the same identifiers, Password and Accept.

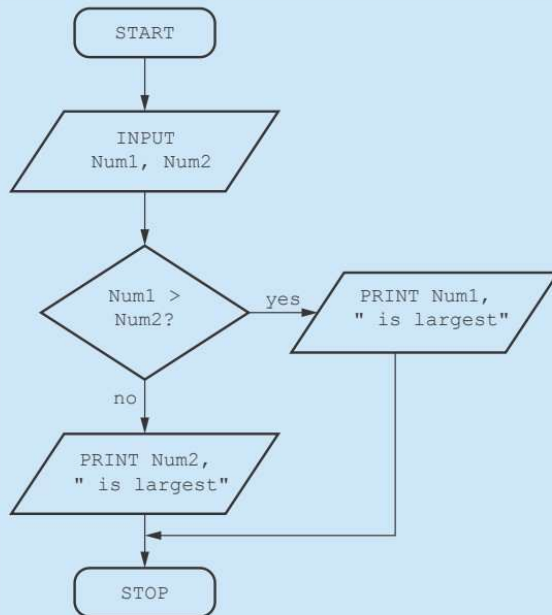
The second answer would gain one mark for part a) since only one other method was identified. For part b)i) the methods are identified, but the terminology used was not accurate, so one mark. For parts b)ii) and iii) code is used for the actual solution and the student has gone on to write a solution in Python that did not exactly match the flowchart. A maximum of two marks could be awarded.

7.3 Explaining the purpose of an algorithm

REVISED

An algorithm sets out the steps to complete a given task. This is usually shown as a flowchart or pseudocode, so that the purpose of the task and the processes needed to complete it are clear to those who study it.

Worked example



```
INPUT Num1, Num2
```

```
IF Num1 > Num2
```

```
    THEN PRINT NUM1, " is largest"
```

```
    ELSE PRINT NUM2, " is largest"
```

```
ENDIF
```

Purpose

To input two numbers, divide the larger number by the smaller number and output the result.

7.4 Standard methods of solution

REVISED

You need to be able to use and understand these standard methods used in algorithms.

Totalling

Sample pseudocode	Notes
<code>Total ← 0</code>	Totalling adds up all the marks for students in the class. When totalling, the total is increased by the value of the mark. Always set the total to zero before checking the mark input.
<code>FOR Count ← 1 TO ClassSize</code>	
<code>INPUT Mark</code>	
<code>Total ← Total + Mark</code>	
<code>NEXT Count</code>	

Counting

Sample pseudocode	Notes
<code>PassCount ← 0</code>	Counting counts the number of students whose mark is over 50. When counting the counter is always increased by 1.
<code>FOR Counter ← 1 TO ClassSize</code>	
<code>INPUT Mark</code>	
<code>IF Mark > 50</code>	
<code>THEN</code>	
<code>PassCount ← PassCount + 1</code>	
<code>ENDIF</code>	
<code>NEXT Counter</code>	

Finding maximum, minimum, average

Sample pseudocode	Notes
<pre> Total ← 0 MaxMark ← 0 MinMark ← 100 FOR Count ← 1 TO ClassSize INPUT Mark IF Mark > MaxMark THEN MaxMark ← Mark ENDIF IF Mark < MinMark THEN MinMark ← Mark ENDIF Total ← Total + Mark NEXT Count Average ← Total / ClassSize </pre>	<p>To find the maximum and minimum values every value must be checked. To find the average all values must be totalled then divided by the number of values.</p> <p>Always set the minimum value to the largest value possible, always set the maximum value to the lowest value possible, before checking the mark input.</p>

Linear search

Sample pseudocode	Notes
<pre> OUTPUT "Enter name to find " INPUT Name Found ← FALSE Counter ← 1 REPEAT IF Name = Name[Counter] THEN Found ← TRUE ELSE Counter ← Counter + 1 ENDIF UNTIL Found OR Counter > ClassSize IF Found THEN OUTPUT Name, " found" ELSE OUTPUT Name, " not found." ENDIF </pre>	<p>A linear search inspects each name in a list in turn to see if the name matches the name that was input.</p> <p>The search stops as soon as the name is found.</p>

Bubble sort

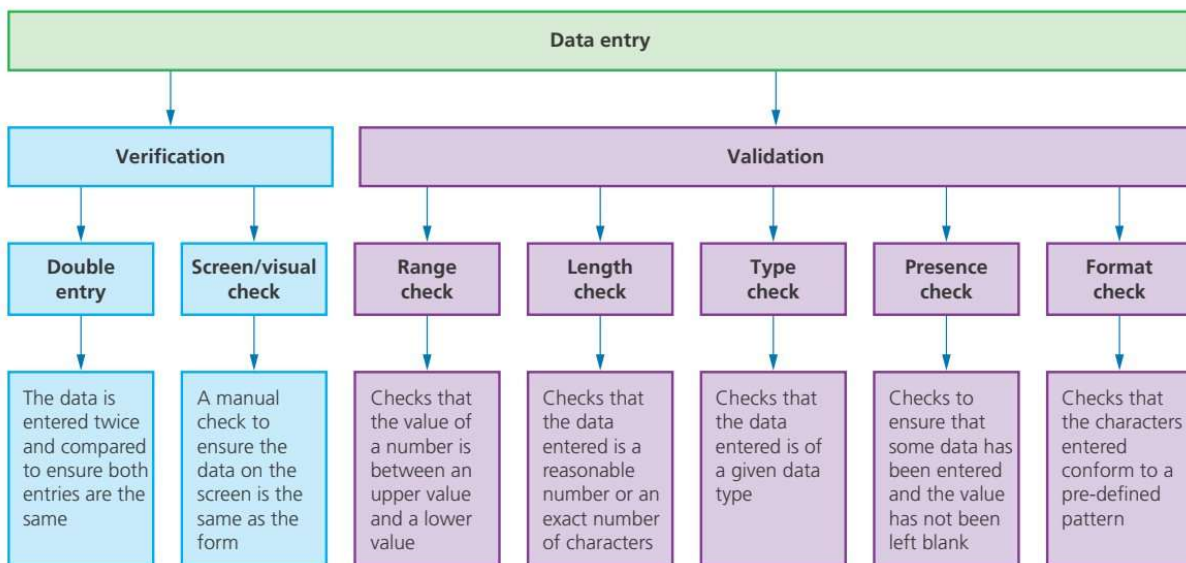
Sample pseudocode	Notes
<pre> First ← 1 Last ← ClassSize REPEAT Swap ← FALSE FOR Index ← First TO Last - 1 IF Name[Index] > Name[Index + 1] THEN Temp ← Name[Index] Name[Index] ← Name[Index + 1] Name[Index + 1] ← Temp Swap ← TRUE ENDIF NEXT Index Last ← Last - 1 UNTIL (NOT Swap) OR Last = 1 </pre>	<p>A bubble sort compares each name in the list is with the next name and swaps them if the names are in the wrong order, starting from the first name and finishing with the next to last name.</p> <p>The last name is in the right place once the list has been checked, this is repeated, decreasing the size of list by one, until no names are swapped or there are no names left to be swapped.</p>

7.5 Validation and verification

REVISED

For data entry, **validation** ensures that only data that is reasonable is accepted. **Verification** is used to check that the data does not change as it is being entered. Different types of checks may be used on the same piece of data.

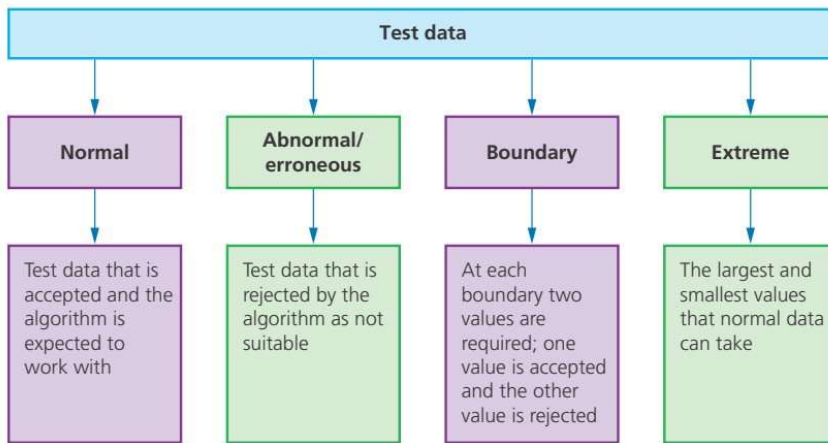
For check digits, please see Section 2.2.3 on page 21, where they are covered more fully.



7.6 Test data

REVISED

Test data ensures that an algorithm works as expected, completing all parts of the solution with no errors. It is used with trace tables to check pseudocode and flowcharts work as expected. **Sets of test data** are used with programs to ensure outputs are as expected.



7.7 Trace tables

REVISED

A trace table records the results from each step in an algorithm; it shows the value of each variable every time that it changes. Working through an algorithm step by step is called a dry run. A trace table is set up with a column for each variable and a column for any output. For example:

A	B	C	X	OUTPUT
0	0	100		

Test data is used to dry run the flowchart and record the results in the trace table.

7.8 Identifying errors in algorithms

REVISED

Trace tables and test data can be used to identify errors and correct errors.

Sample questions and answers

REVISED

Look at this algorithm.

```

01 Small ← 0
02 Counter ← 0
03 REPEAT
04   INPUT Num
05   IF Num < Small
06     THEN
07       Small ← Num
08   ENDIF
09   Counter ← Counter + 1
10 UNTIL Counter ← 10
11 INPUT "Smallest number is ", Small
  
```

It is illegal to photocopy this page

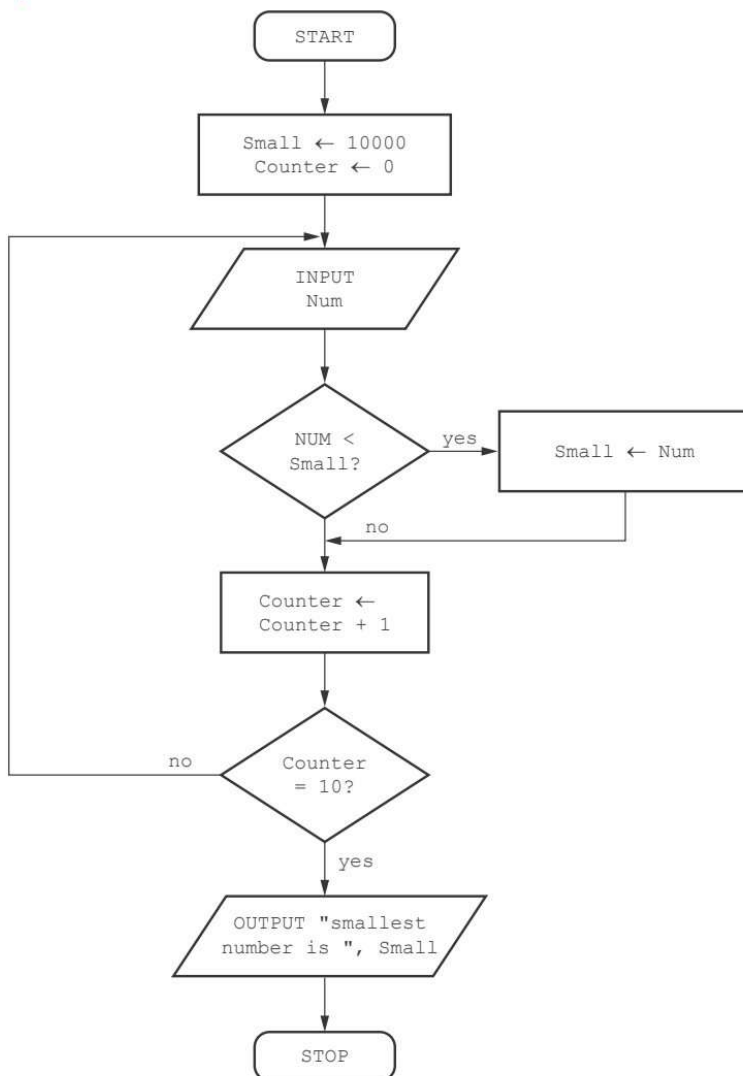
- Find the **three** errors in the algorithm and suggest a correction for each error.
- Write the corrected algorithm as a flowchart.
- Describe the purpose of the corrected algorithm.
- Complete the trace table for the corrected algorithm using this input data:
7, 11, 3, 9, 6, 8, 4, 2, -3, 13, -1, -99
- Extend the pseudocode algorithm to validate that data input is positive.
- Write an extended pseudocode algorithm to find the average of the data input.

[18]

Sample high-level answer

- 01 should be **Small** $\leftarrow$ 10000
10 should be **UNTIL** Counter = 10
11 should be **OUTPUT** Small

b)



Tips

Part a) of this question asks you to identify the errors and provide a corrections; there will probably be only one mark for this. You need to ensure that each error is clearly identified, quoting the line number if given in the question can help.

Part b) requires a flowchart. For full marks you will need to have the correct structure and the correct content. For the structure, you must use the correct flowchart symbols as given in the syllabus, ensure that each flowline has an arrow and clearly label the flowlines out of a decision.

In part c) for the purpose, you answer must state exactly what the algorithm does.

In part d), the trace table must only contain the data that the algorithm uses; sometimes there can be extra values included that are not needed.

In part e) where validation is required, the algorithm must only allow further actions if data that meets the required rules is entered.

In part f), any pseudocode written should be correctly indented and have meaningful identifier names so the intended logic of the algorithm written can be easily understood by the examiner reading the answer.

- c) Input ten numbers, find and output the smallest number input.
d)

Counter	Num	Small	OUTPUT
0		10000	
1	7	7	
2	11		
3	3	3	
4	9		
5	6		
6	8		
7	4		
8	2	2	
9	-3	-3	
10	13		smallest number is -3

- e) Replace line 4 with:

```
04 REPEAT
05 INPUT Num
06 UNTIL Num > 0
```

- f)

```
01 Small ← 10000
02 Counter ← 0
03 Total ← 0
04 REPEAT
05   REPEAT
06     INPUT Num
07   UNTIL Num > 0
08   IF Num < Small
09     THEN
10       Small ← Num
11   ENDIF
12   Total ← Total + Num
13   Counter ← Counter + 1
14 UNTIL Counter = 10
15 Average ← Total / Counter
16 OUTPUT "Smallest number is ", Small, " average is",
    Average
```

Teacher's comments

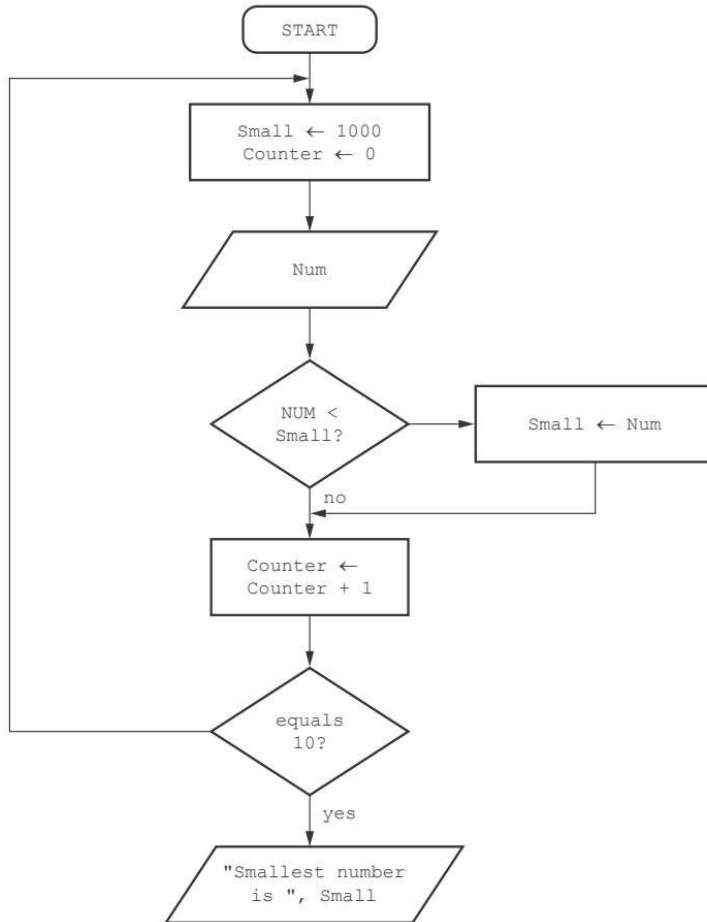
The first answer would gain full marks since in part a) all the errors have clearly identified and corrected, quoting the line number of the error. The student has drawn a flowchart showing the corrected algorithm for part b), it has the correct structure and the correct content. Part c) the student states exactly what the algorithm does. Part d) the trace table contains the data that the algorithm uses, and the correct output.

Part e) the input is validated, and the algorithm only allows further actions if data that meets the required rules has been entered. An error message could have been included. Part f) the pseudocode written is accurate and readable; the only improvement could be to add comments to explain the functionality of the pseudocode.

Sample low-level answer

- a) **Small** $\leftarrow$ should be **Small** $\leftarrow$ 1000
 $\leftarrow$ 10 should be = 10
INPUT should be **OUTPUT**

b)



c) Find the smallest number.

d)

Counter	Num	Small	OUTPUT
		1000	
1	7	7	
2	11		
3	3	3	
4	9		
5	6		
6	8		
7	4		
8	2	2	
9	-3	-3	
10	13		
	-1		
	-99	-99	"Smallest number is -99"

- e) 04 **INPUT** Num
 05 **IF** Num < 0 **THEN** **OUTPUT** "Not positive"

Teacher's comments

The second answer would gain one mark for part a) since only one error has been uniquely identified and corrected. The errors in the assignment and input could be in two places as the line number has not been quoted and there is insufficient code quoted to uniquely identify the position. The flowchart for part b) has a STOP missing, not all flowlines are clearly labelled, and it is unclear where the variables are being input or output. There is also an error in the logic as the values are reinitialised every iteration of the loop. Part c) the answer does not state exactly what the algorithm does, the number of values is missing and the need for input is also missing. Part d) the trace table must only contain the data that the algorithm uses; extra values that are not needed have been included for Num, showing that the algorithm has not been traced, and providing an incorrect output. The output incorrectly has quotation marks round it. Part e) the algorithm still allows further actions if the data entered is negative; only a warning message is output. Part f) the pseudocode written is not correctly indented and does not have meaningful identifier names.

```

f)
01 Small ← 0
02 Counter ← 0
04 REPEAT
05 INPUT Num
06 IF Num < 0 THEN OUTPUT "Not positive"
07 IF Num < Small THEN Small ← Num
08 T ← T + Num
09 Counter ← Counter + 1
10 UNTIL Counter = 10
11 OUTPUT Small, T / Counter

```

7.9 Writing and amending algorithms

REVISED

There are a number of stages when producing an algorithm for a problem.

- 1 Make sure that the problem is clearly specified – the purpose of the algorithm and the tasks to be completed by the algorithm.
- 2 Break the problem down into sub-problems. If it is complex, you may want to consider writing an algorithm for each sub-problem. Most problems, even the simplest ones can be divided into:
 - ☐ set-up processes
 - ☐ permanent storage of data (if required)
 - ☐ input
 - ☐ processing of data
 - ☐ output of results.
- 3 Decide on how any data is to be obtained and stored, what is going to happen to the data and how any results are going to be displayed.
- 4 Design the structure of your algorithm using a structure diagram.
- 5 Decide on how you are going to construct your algorithm, either using a flowchart or pseudocode. If you are told how to construct your algorithm, then follow the guidance.
- 6 Construct your algorithm, making sure that it can be easily read and understood by someone else. Precision is required when writing algorithms, just as it is when writing program code. This involves setting it out clearly and using meaningful names for any data stores.
- 7 Use several sets of test data (normal, abnormal and boundary) to dry run your algorithm and show the results in trace tables, to enable you to find any errors.
- 8 If any errors are found, correct them and repeat the process until you think that your algorithm works perfectly.

Exam-style questions

- 1 Tick (✓) the appropriate column, in the following table, to indicate whether the description applies to validation, verification or neither. You may need to tick more than one column in each row.

[5]

Description	Validation	Verification	Neither
Ensuring a number is positive			
Ensuring a password is as intended			
Ensuring a password is correct			
Ensuring a password contains a number			
Ensuring a password contains a special character			

It is illegal to photocopy this page

- 2 In this question you will be given a statement followed by four possible answers. Select which of the four answers you think is correct and tick (✓) the box next to your answer.
- a) This is a conditional statement:
- ☐ Answer $\leftarrow$ Number1 + Number2
 - ☐ REPEAT ... UNTIL Number1 = Number2
 - ☐ IF Number1 = Number2 THEN ...
 - ☐ WHILE Number1 = Number2 ...
- b) What is meant by the term **decomposition**?
- ☐ Discarding unnecessary details.
 - ☐ Breaking a complex problem into smaller parts.
 - ☐ Showing how a program should be developed.
 - ☐ Writing and testing a program.
- c) What is meant by the term **abstraction**?
- ☐ Identifying data that needs to be stored.
 - ☐ Breaking sub-systems down into smaller sub-systems.
 - ☐ Keeping the key elements and discarding information not needed.
 - ☐ Stage of the program development life cycle. [3]
- 3 a) Write pseudocode to input 12 numbers and store them in an array. [3]
 b) Change your pseudocode to use a different loop structure.
 c) Identify another loop structure you could have used.
 d) Write pseudocode to find the largest, smallest and the average of the numbers you have stored.
 e) Write pseudocode to sort the numbers you have stored in descending order. [16]
- 4 Explain the difference between mathematical and comparison operators. Include an example of each type in your answer. [4]
- 5 Identify **three** types of test data. Include an example of each type in your answer with an explanation of why it is appropriate. [9]
- 6 Identify **four** errors in this algorithm. [4]

```

01 NumberProducts  $\leftarrow$  50 // length of array ProductName[]
02 OUTPUT "Please enter product to find "
03 INPUT Product
04 Found  $\leftarrow$  FALSE
05 Counter  $\leftarrow$  1
06 REPEAT
07   IF Product = ProductName[Counter]
08     THEN
09       Found  $\leftarrow$  FALSE
10     ELSE
11       Counter  $\leftarrow$  Counter - 1
12   ENDIF
13 UNTIL Found AND Counter > NumberProducts
14 IF Found
15   THEN
16     OUTPUT Product, " found at position ", Counter, " in the list."
17   ELSE
18     OUTPUT Name, " not found."
19   ENDIF

```

8

Programming

Key objectives

The objectives of this chapter are to revise:

- programming concepts:
 - use of variables, constants and data types
 - input and output
 - sequence
 - selection including nesting
 - iteration
 - totalling and counting
 - string handling
 - operators – arithmetic, logical and Boolean
- procedures and functions including the use of:
 - parameters
 - local and global variables
 - library routines
 - creating a maintainable program
- arrays: including one- and two-dimensional arrays, use of indexes, use of iteration for reading from and writing to arrays
- file handling: including opening, closing, reading from and writing to data and text files

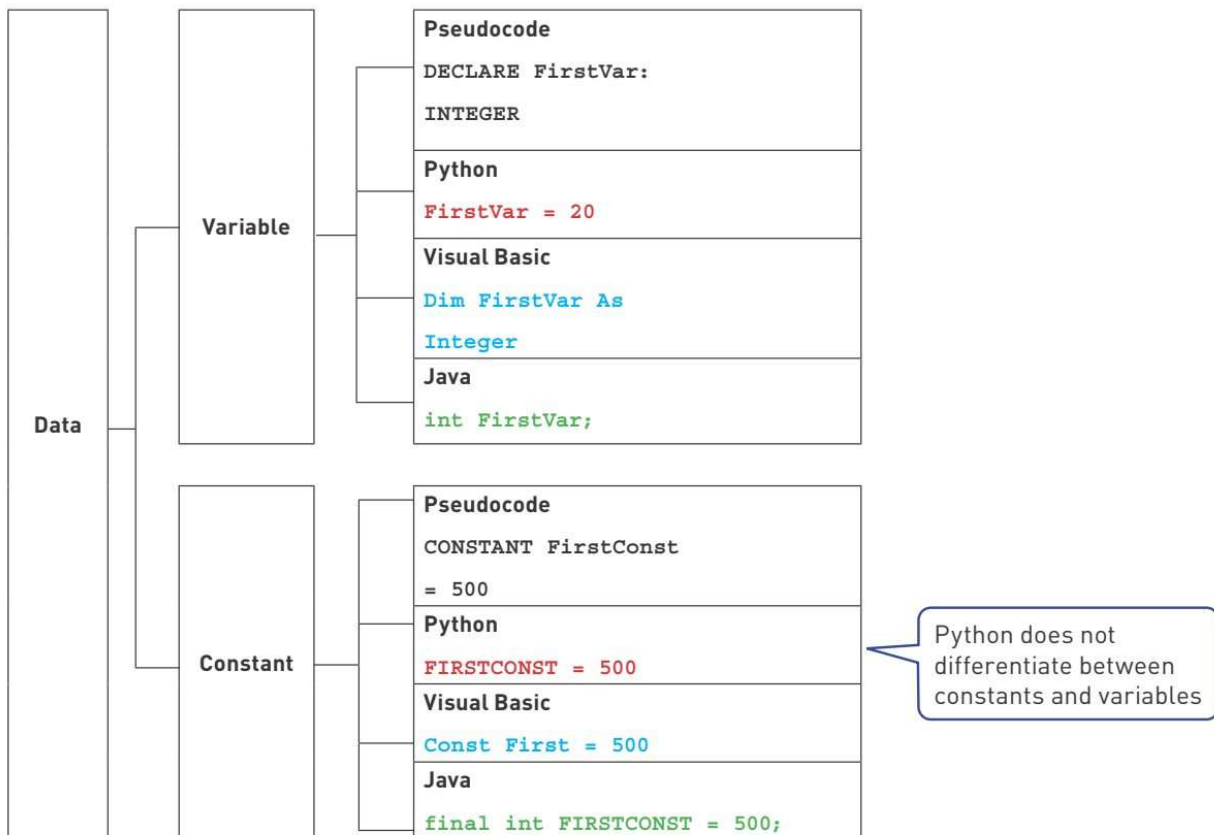
8.1 Programming concepts

REVISED

8.1.1 Variables and constants

Variables and **constants** refer to a data store containing a single value, identified by a meaningful name. Variables may change during the execution of a program, but constants do not change. Constants and variables should be **declared** before they are used in the program.

The diagram below shows examples of how constants and variables are used.



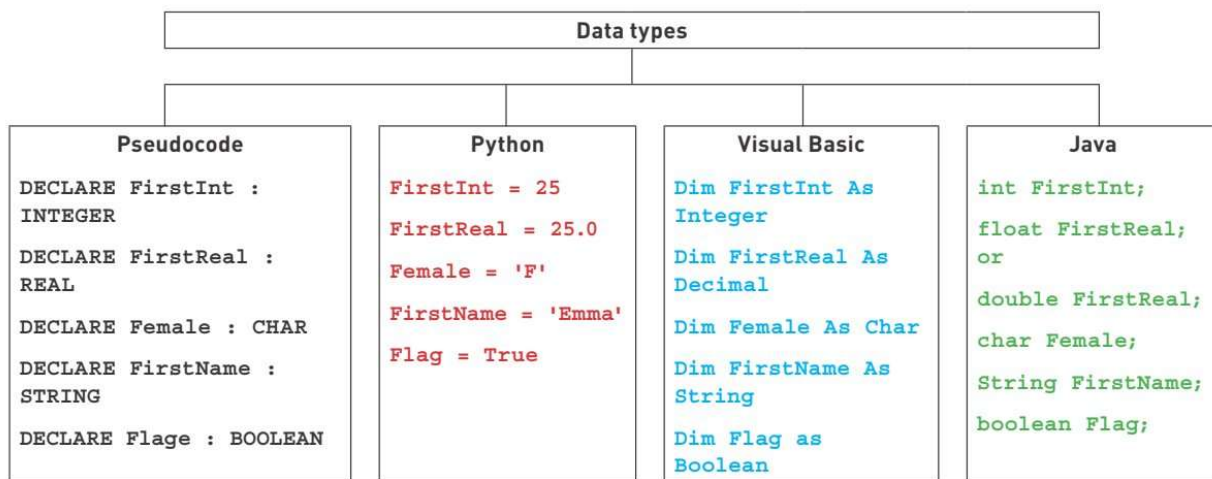
It is illegal to photocopy this page

8.1.2 Basic data types

For a computer system to process and store data effectively, different kinds of data are given different types.

- **Integer** – a positive or negative whole number that can be used with mathematical operators.
- **Real** – a positive or negative number with a fractional part; real numbers can be used with mathematical operators.
- **Char** – a variable or constant that is a single character.
- **String** – a variable or constant that is several characters in length.
- **Boolean** – a variable or constant that can have only two values, TRUE or FALSE.

If a number is stored as a string or char then it cannot be used in calculations. The chart below shows examples of data types.



8.1.3 Input and output

To enter data and output results, programs need to use input and output statements. You need to write algorithms and programs that take input from a keyboard and output to a screen. For a program to be useful, the user needs to know what they are expected to input, so each input needs to be accompanied by a **prompt** stating the input required. Here are some examples of input statements with prompts.

Pseudocode <pre> OUTPUT "Enter a whole number " INPUT FirstInt </pre>
Python <pre> FirstInt = int(input ("Enter a whole number")) </pre>
Visual Basic <pre> Console.Write("Enter a whole number ") FirstInt = Integer.Parse (Console.ReadLine()) </pre>
Java <pre> System.out.println("Enter a whole number "); int FirstInt = myObj.nextInt(); </pre>

Here are some examples of outputs that include suitable **messages** as well as values.

Pseudocode
OUTPUT "Number is " FirstInt
Python
Print ("Number is ", FirstInt)
Visual Basic
Console.WriteLine("Number is " & FirstInt)
Java
System.out.println("Number is " + FirstInt);

8.1.4 Basic concepts

These are the concepts to be used and understood when writing the steps needed to solve a problem.

Sequence

The ordering of the steps in an algorithm. An incorrect order can lead to incorrect results or unnecessary extra steps. For example, a variable used to store a running total must be set to zero before use otherwise the final total could be incorrect.

Selection

This allows different routes through the steps of a program. Case statements are used when there are multiple choices that can be made.

IF statement examples	Language
<pre>IF Age > 17 THEN OUTPUT "You are an adult" ELSE OUTPUT "You are a child" ENDIF</pre>	Pseudocode
<pre>if Age > 17: print "You are an adult" else: print "You are a child"</pre>	Python uses else with a colon, : and indentation:
<pre>If Age > 17 Then Console.WriteLine("You are an adult") Else Console.WriteLine("You are a child") End If</pre>	Visual Basic uses Else and End If
<pre>If (Age > 17) { System.out.println ("You are an adult"); } else { System.out.println ("You are a child"); }</pre>	Java uses else and curly brackets, { , and uses } instead of ENDIF .

Case statement examples	Language
<pre> CASE OF OpValue "+" : Answer = Number1 + Number2 "-" : Answer = Number1 - Number2 "*" : Answer = Number1 * Number2 "/" : Answer = Number1 / Number2 OTHERWISE OUTPUT "invalid operator") ENDCASE </pre>	Pseudocode
<pre> if OpValue == "+": Answer = Number1 + Number2 elif OpValue == "-": Answer = Number1 - Number2 elif OpValue == "*": Answer = Number1 * Number2 elif OpValue == "/": Answer = Number1 / Number2 else: print("invalid operator") </pre>	Python uses elif for multiple tests
<pre> Select Case OpValue Case "+" Answer = Number1 + Number2 Case "-" Answer = Number1 - Number2 Case "*" Answer = Number1 * Number2 Case "/" Answer = Number1 / Number2 Case Else Console.WriteLine("invalid operator") End Select </pre>	Visual Basic uses Select Case and Case Else instead of CASE and OTHERWISE
<pre> switch (OpValue) { case "+": Answer = Number1 + Number2; break; case "-": Answer = Number1 - Number2; break; case "*": Answer = Number1 * Number2; break; case "/": Answer = Number1 / Number2; break; default: System.out.println("invalid operator"); } </pre>	Java uses default instead of OTHERWISE and uses break to pass control to the end of the code block when a section is finished.

Iteration

There are three types of loop structure which perform iterations to repeat programming code.

- count-controlled loops (for a set number of iterations)
- pre-condition loops – may have no iterations
- post-condition loops – always has at least one iteration.

For statement examples	Language
<pre>FOR Counter ← 1 TO 10 STEP 2 OUTPUT Counter NEXT Counter</pre>	Pseudocode
<pre>for Counter in range (1,10,2): print(Counter)</pre>	Python uses the range function, a colon to show the start of the for loop and indentation of all statements in the for loop.
<pre>For Counter = 1 To 10 Step 2 Console.WriteLine(Counter) Next</pre>	Visual Basic uses Step and Next
<pre>for (int Counter = 1; Counter <= 10; Counter = Counter + 2) { System.out.println(Counter); }</pre>	Java uses { } to show the start and end of the for loop.
Pre-condition loop examples	Language
<pre>WHILE TotalWeight < 100 TotalWeight = TotalWeight + Weight ENDWHILE</pre>	Pseudocode
<pre>while TotalWeight < 100: TotalWeight = TotalWeight + Weight</pre>	Python uses a colon to show the start of the while loop and indentation to show which statements are in the while loop.
<pre>While TotalWeight < 100 TotalWeight = TotalWeight + Weight End While</pre>	Visual Basic uses While and End While
<pre>while (TotalWeight < 100) { TotalWeight = TotalWeight + Weight; }</pre>	Java uses { } to show the start and end of the while loop.
Post-condition loop examples	Language
<pre>REPEAT NumberOfItems = NumberOfItems + 1 UNTIL NumberOfItems > 19</pre>	Pseudocode
	Python only uses pre-condition loops.
<pre>Do NumberOfItems = NumberOfItems + 1 Loop Until NumberOfItems > 19</pre>	Visual Basic uses Do and Loop Until
<pre>do { NumberOfItems ++; } while (NumberOfItems <= 20);</pre>	Java uses do and while so the condition is the opposite of the until condition used in Visual Basic. Look at the code carefully to see the difference. Java uses { } to show the start and end of the loop.

Totalling and counting

Totalling and **counting** are standard methods.

See Chapter 7 for more on totalling and counting.

Totalling examples	Counting examples	Language
<code>TotalWeight = TotalWeight + Weight</code>	<code>NumberOfItems = NumberOfItems + 1</code>	Pseudocode
<code>TotalWeight = TotalWeight + Weight</code>	<code>NumberOfItems = NumberOfItems + 1</code>	Python
<code>TotalWeight = TotalWeight + Weight</code>	<code>NumberOfItems = NumberOfItems + 1</code>	Visual Basic
<code>TotalWeight = TotalWeight + Weight;</code>	<code>NumberOfItems ++;</code>	Java

String handling

Strings store text: every string contains several characters and each character in a string can be found by its position number. The first character of a string in pseudocode is at position 1; the first character of a string is at position zero for Java, Python and Visual Basic.

You need to know these methods:

- **Length** – finding the number of characters in the string. For example, the length of the string "Computer Science" is 16 characters as spaces are counted as a character.
- **Substring** – extracting part of a string. For example, the substring "Science" could be extracted from "Computer Science".
- **Upper** – converting all the letters in a string to uppercase. For example, the string "Computer Science" would become "COMPUTER SCIENCE".
- **Lower** – converting all the letters in a string to lowercase. For example, the string "Computer Science" would become "computer science".

Length	Language and notes
<code>LENGTH("Computer Science")</code>	Pseudocode Text in quotes can be used or a variable with data type string.
<code>LENGTH(MyString)</code>	
<code>len("Computer Science")</code>	Python Text in quotes can be used or a variable with data type string.
<code>len(MyString)</code>	
<code>"Computer Science".Length()</code>	Visual Basic Text in quotes can be used or a variable with data type string.
<code>MyString.Length()</code>	
<code>"Computer Science".Length();</code>	Java Text in quotes can be used or a variable with data type string.
<code>MyString.length();</code>	
Substring – extracts 'Science'	Language and notes
<code>SUBSTRING("Computer Science", 10, 7)</code>	Pseudocode Text in quotes can be used or a variable with data type string. First parameter is the string, second parameter is the position of the start character, third parameter is the length of the required substring. Pseudocode strings start at position one.
<code>SUBSTRING(MyString, 10, 7)</code>	
<code>"Computer Science"[9:16]</code>	Python Text in quotes can be used or a variable with data type string. Strings are treated as lists of characters in Python. First index is the start position of the substring, second index is the end position of the substring. Python strings start at position zero.
<code>MyString[9:16]</code>	
<code>"Computer Science".Substring(9, 7)</code>	Visual Basic Text in quotes can be used or a variable with data type string. First parameter is the start position of the substring, second parameter is the length of the substring. Visual Basic strings start at position zero.
<code>MyString.Substring(9, 7)</code>	
<code>"Computer Science".substring(9,17);</code>	Java Text in quotes can be used or a variable with data type string. First parameter is the start position of the substring, second parameter is the exclusive end position of the substring. Java strings start at position zero.

Upper	Language and notes
<code>UCASE("Computer Science")</code> <code>UCASE(MyString)</code>	Pseudocode Text in quotes can be used or a variable with data type string.
<code>"Computer Science".upper()</code> <code>MyString.upper()</code>	Python Text in quotes can be used or a variable with data type string.
<code>UCase("Computer Science")</code> <code>UCase(MyString)</code>	Visual Basic Text in quotes can be used or a variable with data type string.
<code>"Computer Science".toUpperCase();</code> <code>MyString.toUpperCase();</code>	Java Text in quotes can be used or a variable with data type string.
Lower	Language and notes
<code>LCASE("Computer Science")</code> <code>LCASE(MyString)</code>	Pseudocode Text in quotes can be used or a variable with data type string.
<code>"Computer Science".lower()</code> <code>MyString.lower()</code>	Python Text in quotes can be used or a variable with data type string.
<code>LCase("Computer Science")</code> <code>LCase(MyString)</code>	Visual Basic Text in quotes can be used or a variable with data type string.
<code>"Computer Science".toLowerCase();</code> <code>MyString.toLowerCase();</code>	Java Text in quotes can be used or a variable with data type string.

Arithmetic, logical and Boolean operators

There are three types of operator:

- **arithmetic operators** – for calculations (+, -, *, /, ^, MOD, DIV)
- **logical** – for comparisons
- **Boolean** – to decide whether an expression is true or false.

Arithmetic	Pseudocode	Python	Visual Basic	Java
Add	+	+	+	+
Subtract	-	-	-	-
Multiply	*	*	*	*
Divide	/	/	/	/
Raise to the power of	^	**	^	<code>import java.lang.Math;</code> <code>Math.pow(x, y);</code>
Group	()	()	()	()
Remainder division	MOD	<i>For more on these see section 8.1.7 Library routines.</i>		
Integer division	DIV			
Logical	Pseudocode	Python	Visual Basic	Java
Greater than	>	>	>	>
Less than	<	<	<	<
Equal	=	==	=	==
Greater than or equal	>=	>=	>=	>=
Less than or equal	<=	<=	<=	<=
Not equal	<>	!=	<>	!=
Boolean	Pseudocode	Python	Visual Basic	Java
Both True	AND	and	And	&&
Either True	OR	or	Or	
Not True	NOT	Not	Not	!

Sample questions and answers

REVISED

A programmer has been asked to design and write a program to accept inputs about each game player:

- name
- age in years
- nickname
- if the player wants to use their name or their nickname.

The program must include a validation check on the age, ensure the name and the nickname are no more than eight characters long, then output a personalised welcome message for each player.

There can be three, four or five players for each game; the input routine finishes when all the players have entered their details.

- a) Give a suitable identifier name and data type for each input. Explain your choice of data type for each input.
- b) Write your design for the program in pseudocode. Include the validation checks for input. Your program must include selection and iteration, use comments to show where these have been used and give the reason why they have been used.

[12]

Tips

Part a) of this question asks for the name and the data type of each variable used to store the data input. This means your answer must give a meaningful name for each variable and an appropriate data type from those included in the syllabus. Make sure the names are long enough to be meaningful to someone else and they should begin with a capital letter for example **Age** not **A** or **age**. Also, the data type must clearly belong to that variable, and you need to say why the data type is appropriate.

Part b) requires a complete algorithm written in Pseudocode, not program code, including selection, an IF statement, and iteration, a loop, to provide a design for every step required. This includes declaration and initialisation of the variables required, input, validation and output of a welcome message for every player using their name or nickname as required. Comments must be included for the iteration and selection. To make the algorithm readable suitable indentation must be used.

Teacher's comments

This first answer is clearly set out and readable. The use of a table for part a) ensures that each variable has all the appropriate information.

For part b)

- all the variables used have been declared with meaningful names and initialised if required
- indentation and comments have been used sensibly
- both iteration and selection have been used
- all the inputs have clear prompts
- the validation checks ensure re-entry of data that does not meet a check
- the welcome output only includes the name or the nickname as chosen
- between three and five players can be entered.

Sample high-level answer

a)

Identifier	Data type	Reason
Name	String	eight characters or less is required
Age	Integer	a whole number that can be validated with a range check is required
Nickname	String	eight characters or less is required
Choice	Boolean	two choices only required

```

b) DECLARE Name : STRING
   DECLARE Age : INTEGER
   DECLARE Nickname : STRING
   DECLARE Choice, Another : BOOLEAN
   DECLARE Player : INTEGER
   Player ← 0
   Another ← TRUE
   REPEAT // Iteration to enter at least 3 players
       REPEAT
           OUTPUT "Enter your name, use 8 characters or less "
           INPUT Name
       UNTIL LENGTH(Name) <= 8
       REPEAT
           OUTPUT "Enter your age in years "
           INPUT Age
       UNTIL Age > 5 AND Age < 100
       REPEAT
           OUTPUT "Enter your nickname, use 8 characters or less "
           INPUT Nickname
       UNTIL LENGTH(Nickname) <= 8
       OUTPUT "Use your name (True) or Nickname (FALSE) "
       INPUT Choice
       IF Choice // Selection whether to use the name or the nickname
           THEN
               OUTPUT "Welcome ", Name, " aged ", Age
           ELSE
               OUTPUT "Welcome ", Nickname, " aged ", Age
       ENDIF
       Player ← Player + 1
       IF Player >= 3 AND Player < 6 // selection for another player
           THEN
               OUTPUT "Do you want to add another player TRUE or FALSE? "
               INPUT Another
           ENDIF
       UNTIL Player = 6 OR NOT Another

```

Sample low-level answer

a) My variables are N1, N2 and A, they are all text because there are no calculations.

b)

```

INPUT N1
IF LENGTH(N1) > 8 THEN OUTPUT "Too long"
INPUT A
INPUT N2
IF LENGTH(N2) > 8 THEN OUTPUT "Too long"
IF Age > 100 THEN OUTPUT "Too old"
OUTPUT "Welcome ", N1, N2 " aged ", A
INPUT N1
IF LENGTH(N1) > 8 THEN OUTPUT "Too long"
INPUT A
INPUT N2
IF LENGTH(N2) > 8 THEN OUTPUT "Too long"
IF Age > 100 THEN OUTPUT "Too old"
OUTPUT "Welcome ", N1, N2 " aged ", A
INPUT N1
IF LENGTH(N1) > 8 THEN OUTPUT "Too long"
INPUT A
INPUT N2
IF LENGTH(N2) > 8 THEN OUTPUT "Too long"
IF Age > 100 THEN OUTPUT "Too old"
OUTPUT "Welcome ", N1, N2 " aged ", A
INPUT N1
IF LENGTH(N1) > 8 THEN OUTPUT "Too long"
INPUT A
INPUT N2
IF LENGTH(N2) > 8 THEN OUTPUT "Too long"
IF Age > 100 THEN OUTPUT "Too old"
OUTPUT "Welcome ", N1, N2 " aged ", A

```

Teacher's comments

This second answer is poorly set out and not easy to understand. Only three variables are identified in part a) and the names are too short to be meaningful to another reader, the single data type is not from the syllabus and the reason is weak.

For part b)

- none the variables used have been declared
- indentation and comments are missing
- only selection has been used; code is repeated instead of using iteration
- inputs have no prompts, one is missing
- the validation checks only give a warning message
- the welcome output includes both the name and the nickname
- only five players can be entered.

8.1.5 Use of nested statements

Selection and iteration statements can be nested one inside another. One type of construct can be nested within another, for example selection can be nested within a condition-controlled loop, or a loop can be nested within a selection statement.

8.1.6 Procedures and functions

There are often similar tasks to perform that make use of the same groups of statements. Many programming languages make use of subroutines, also known as named **procedures** or **functions**, instead of repeating these statements.

- A procedure is a set of programming statements grouped together under a single name that can be called to perform a task at any point in a program.
- A function is a set of programming statements grouped together under a single name that can be called to perform a task at any point in a program. A function returns a value back to the main program.
- **Parameters** are the variables that store the values of the arguments passed to a procedure or function. Some procedures and functions will have parameters.

Procedures and functions are defined once and can be called many times within a program.

Here is an example of a procedure without parameters. Different terminology is used by each programming language for 'procedure' – void functions (Python), subroutines (Visual Basic), methods (Java).

Procedure Stars – definition	Call	Language
<pre>PROCEDURE Stars OUTPUT "*****" ENDPROCEDURE</pre>	Stars	Pseudocode
<pre>def Stars(): print("*****")</pre>	Stars()	Python
<pre>Sub Stars() Console.WriteLine("*****") End Sub</pre>	Stars()	Visual Basic
<pre>static void Stars() { System.out.println("*****"); }</pre>	Stars();	Java

Here is an example of how a procedure with parameters can be defined. For IGCSE Computer Science the number of parameters used is limited to two.

Procedure Stars with parameter – definition	Call	Language
<pre>PROCEDURE Stars (Number:INTEGER) DECLARE Counter : INTEGER FOR Counter ← 1 TO Number OUTPUT "*" NEXT ENDPROCEDURE</pre>	Stars(7)	Pseudocode Counter is a local variable.
<pre>def Stars(Number): for counter in range (Number): print("{}", end = '')</pre>	Stars(7)	Python Note: <code>end = ''</code> ensures that the stars are printed on one line without spaces between them.
<pre>Sub Stars(Number As Integer) Dim Counter As Integer For Counter = 1 To Number Console.Write(" ") Next End Sub</pre>	Stars(7)	Visual Basic Counter is a local variable.
<pre>static void Stars(int Number) { for (int Counter = 1; Counter <= Number; Counter ++)</pre> <pre>{ System.out.print(" "); }</pre> <pre>}</pre>	Stars(7);	Java Counter is a local variable.

A function is like a procedure, except it always returns a value. Like a procedure, it is defined once and can be called many times within a program, and be defined with or without parameters. Different terminology is used for 'function' by some programming languages: fruitful functions (Python), functions (Visual Basic), methods with returns (Java).

The first statement in a procedure or function is called the **header**. It always contains the name, it may contain the parameters required and it may contain the data type of the return value for a function.

For IGCSE Computer Science the number of parameters used in a function or procedure definition and the **arguments** in the call are limited to two.

There must always be a return statement in a procedure definition to specify the value to be returned.

Here is a function to convert temperature from Fahrenheit to Celsius.

Function with a parameter – temperature conversion example		
Definition	Call	Language
<pre>FUNCTION Celsius(Temperature: REAL) RETURNS REAL RETURN (Temperature - 32) / 1.8 ENDFUNCTION</pre>	<pre>MyTemp ← Celsius(MyTemp)</pre>	Pseudocode
<pre>def Celsius(Temperature): return (Temperature - 32) / 1.8</pre>	<pre>MyTemp = Celsius(MyTemp)</pre>	Python Data type of function does not need to be defined.
<pre>Function Celsius(ByVal Temperature As Decimal) As Decimal Return (Temperature - 32) / 1.8 End Function</pre>	<pre>MyTemp = Celsius(MyTemp)</pre>	Visual Basic
<pre>static double Celsius(double Temperature) { return (Temperature - 32) / 1.8; }</pre>	<pre>MyTemp = Celsius(MyTemp)</pre>	Java

Local and global variables

A **global variable** can be used in any part of a program – its **scope** covers the whole program.

A **local variable** can only be used by the part of the program it has been declared in; it is restricted to that part of the program.

8.1.7 Library routines

Library routines are fully tested and ready for use in a program. An IDE includes a standard library of functions and procedures. Examples are:

- **MOD** – returns remainder of a division
- **DIV** – returns the quotient (that is, the whole number part) of a division
- **ROUND** – returns a value rounded to a given number of decimal places
- **RANDOM** – returns a random number.

Here are some examples of these library routines in pseudocode and each programming language.

Examples for MOD, DIV, ROUND and RANDOM	Language
<pre>Value1 ← MOD(10, 3) Value2 ← DIV(10, 3) Value3 ← ROUND(6.97354, 2) Value4 ← RANDOM()</pre>	Pseudocode
<pre>Value1 = 10%3 Value2 = 10//3 Value = divmod(10,3) Value3 = round(6.97354, 2) from random import random Value4 = random()</pre>	Python MOD uses the % operator and DIV the // operator. The function divmod(x,y) provides both answers where the first answer is DIV and the second answer is MOD. RANDOM needs to import the library routine random
<pre>Value1 = 10 Mod 3 Value2 = 10 \ 3 Value3 = Math.Round(6.97354, 2) Value4 = Rnd()</pre>	Visual Basic DIV uses the \ operator.

Examples for MOD, DIV, ROUND and RANDOM	Language
<pre>import java.lang.Math; Value1 = 10%3; Value2 = 10/3; Value3 = Math.round(6.97354 * 100)/100.0; import java.util.Random; Random rand = new Random(); double Value4 = rand.nextDouble();</pre>	<p>Java</p> <p>MOD uses the % operator.</p> <p>DIV uses the normal division operator; if the numbers being divided are both integers then integer division is performed, as shown.</p> <p>Java imports the library routine Math</p> <p>Math.round only rounds to whole numbers.</p> <p>RANDOM needs to import the library routine Random</p>

8.1.8 Creating a maintainable program

A program should be understandable to another programmer. A maintainable program should:

- always use meaningful identifier names for variables, constants, arrays, procedures and functions
- be divided into modules for each task using procedures and functions
- be fully commented using your programming language's commenting feature.

Here are programming languages commenting features.

Example comments	Language
// pseudocode uses a double slash to start a comment	Pseudocode
#Python uses hash to start a comment for every line	Python
'Visual Basic uses a single quote to start a comment 'for every line	Visual Basic
// Java uses a double slash to start a single line comment and /* to start multiple line comments and to end them */	Java

8.2 Arrays

REVISED

An **array** is a data structure containing elements of the same data type. The position of each element is identified using the array's **index** and the array name. Arrays are used for storing similar items of data, for example a list names of students in a class or a table of hourly weather station readings for a week.

The first element of an array can have an index of zero or one. Most programming languages automatically set the first index of an array to zero.

8.2.1 One- and two- dimensional arrays

Arrays can be one-dimensional or two-dimensional. A one-dimensional array can be referred to as a list, while a two-dimensional array can be referred to as a table.

Index	myList		MyTable			
[0]	27	A list with 10 elements in it. The first element has an index of zero.	27	31	17	A table with 10 rows and 3 columns. The first element is located at position 0,0
[1]	19		19	67	48	
[2]	36		36	98	29	
[3]	42		42	22	95	
[4]	16		16	35	61	
[5]	89		89	46	47	
[6]	21		21	71	28	
[7]	16		16	23	13	
[8]	55		55	11	77	
[9]	72		34	76	21	

Notes on Python and arrays

Instead of arrays, Python uses another object called a list. The difference between arrays and lists are minimal at IGCSE level but the main differences are:

- a list can contain different data types whereas arrays must all hold the same type of data
- to achieve the same structure as a two-dimensional array, Python embeds one list within another.

8.2.2 Declaring and populating arrays with iteration

The table above can be populated using a nested loop, one for each index:

```

DECLARE MyTable : ARRAY[0:9,0:2] OF INTEGER
OUTPUT "Enter these values in order"
OUTPUT "27, 19, 36, 42, 16, 89, 21, 16, 55, 34"
OUTPUT "31, 67, 98, 22, 35, 46, 71, 23, 11, 76"
OUTPUT "17, 48, 29, 95, 61, 47, 28, 13, 77, 21"
FOR ColumnIndex ← 0 TO 2
  FOR RowIndex ← 0 TO 9
    OUTPUT "Enter next value "
    INPUT MyTable[RowIndex, ColumnIndex]
  NEXT RowIndex
NEXT ColumnIndex

```

Sample questions and answers

REVISED

Extend your program design (from page 120) to store each name and nickname in a two-dimensional array, the age in another array and the choice in another array. The index of the array will identify the player. Use a procedure to output the personalised welcome message for each player when all players have entered their details. This output should be in ascending order of age.

- Declare and initialise the arrays needed.
- Write your design for the program in pseudocode. You do not need to include the declarations for the arrays and the validation checks for input.

[12]

It is illegal to photocopy this page

Tips

Part a) of this question asks you to declare and initialise each array used to store the data input. This means your answer must write a declaration with a meaningful name for each array and an appropriate data type from those included in the syllabus. Make sure the arrays are large enough to store details for five players. Also, the array for the name and nickname must be a two-dimensional array.

Part b) requires a complete algorithm written in pseudocode, not program code, including a procedure and a sort. Comments should be included to make the algorithm readable, and suitable indentation must be used.

Sample high-level answer

a)

```

DECLARE NameAndNickname : ARRAY[1:5, 1:2] OF STRING
DECLARE Age : ARRAY[1:5] OF INTEGER
DECLARE Choice : ARRAY[1:5] OF BOOLEAN

```

b)

```

// procedure definition for the welcome messages
PROCEDURE Welcome (PlayerNo : INTEGER)
    DECLARE Count, Last, TempAge: INTEGER
    DECLARE Swap, TempChoice : BOOLEAN
    DECLARE TempString : STRING
    Last ← PlayerNo

    REPEAT // bubble sort all arrays on Age in ascending order
        Swap ← FALSE
        FOR COUNT ← 1 TO Last - 1
            IF Age[Count] > Age[Count + 1]
                THEN
                    TempAge ← Age[Count]
                    Age[Count] ← Age[Count + 1]
                    Age[Count + 1] ← TempAge
                    TempChoice ← Choice [Count]
                    Choice [Count] ← Choice [Count + 1]
                    Choice [Count + 1] ← TempChoice
                    TempString ← NameAndNickname [Count,1]
                    NameAndNickname [Count,1] ← NameAndNickname [Count + 1,1]

```

```

        NameAndNickname [Count + 1,1] ← TempString
        TempString ← NameAndNickname [Count,2]
        NameAndNickname [Count,2] ← NameAndNickname [Count + 1,2]
        NameAndNickname [Count + 1,2] ← TempString
        Swap ← TRUE
    NEXT Count
    Last ← Last - 1
UNTIL (NOT Swap) or Last = 1

FOR Count ← 1 TO PlayerNo // print welcome messages
    IF Choice[Count]
    THEN
        OUTPUT "Welcome ", NameAndNickname [Count,1], " aged ", Age[Count]
    ELSE
        OUTPUT "Welcome ", NameAndNickname [Count,2], " aged ", Age[Count]
    ENDIF
NEXT Count
ENDPROCEDURE

DECLARE Player : INTEGER
DECLARE Another : BOOLEAN
Player ← 1
Another ← TRUE
REPEAT // Iteration to enter at least 3 players
    OUTPUT "Enter your name, use 8 characters or less "
    INPUT NameAndNickname[Player,1]
    OUTPUT "Enter your age in years "
    INPUT Age[Player]
    OUTPUT "Enter your nickname, use 8 characters or less "
    INPUT NameAndNickname[Player,2]
    INPUT Choice[Player]
    Player ← Player + 1
IF Player > 3 AND Player <= 6 // selection for another player
    THEN
        OUTPUT "Do you want to add another player TRUE or FALSE? "
        INPUT Another
    ENDIF
UNTIL Player = 6 OR NOT Another
Player = Player - 1 // setting Player to the number of players entered
Welcome (Player)// procedure call

```

Sample low-level answer

a) My arrays are N1, N2 and A, they are all text because there are no calculations.

b)

```

INPUT N1[1]
INPUT A[1]
INPUT N2[1]
INPUT N1[2]
INPUT A[2]
INPUT N2[2]
INPUT N1[3]
INPUT A[3]
INPUT N2[3]
INPUT N1[4]
INPUT A[4]
INPUT N2[4]
INPUT N1[5]
INPUT A[5]
INPUT N2[5]
N1.sort()
A.sort()
N2.sort()
OUTPUT "Welcome ", N1, N2 " aged ", A

```

Teacher's comments

The first answer is clearly set out and readable. The array declarations for part a) correctly include the two-dimensional array for name and nickname.

For part b)

- local and global variables have been declared with meaningful names and initialised when required
- indentation and comments have been used sensibly
- a procedure has been declared and called
- all the arrays have been sorted on ascending order of age using a bubble sort
- the welcome output only includes the name or the nickname as chosen
- between three and five players can be entered.

The second answer is poorly set out and not easy to understand. Three arrays have been identified but not declared in part a), the names are too short to be meaningful to another reader, and a two-dimensional array for name and nickname has not been used.

For part b)

- none the variables used have been declared
- indentation and comments are missing
- a procedure has not been used
- the three arrays have been sorted independently using a sort library routine
- the welcome output includes both the name and the nickname
- only five players can be entered.

8.3 File handling

REVISED

8.3.1 Purpose of storing data in a file

Computer programs store data that will be required again in a **file**. When data is saved to a file it is stored permanently and can be accessed by the same program at a later date, accessed by another program, or sent to be used on another computer(s).

8.3.2 Using files

Every file is identified by its filename. Files can be created, opened, closed, written to, read from and deleted.

Here is an example of writing a line of text to a file and reading the line of text back from the file. Refer to the *Cambridge IGCSE and O Level Computer Science Second Edition* Student's Book, pages 334 and 335 for examples in other programs.

```

DECLARE TextLine : STRING // variables are declared as normal
DECLARE MyFile : STRING
MyFile ← "MyText.txt"

// writing the line of text to the file
OPEN MyFile FOR WRITE // opens file for writing
    OUTPUT "Please enter a line of text"
    INPUT TextLine
    WRITEFILE, TextLine // writes a line of text to the file
CLOSEFILE(MyFile) // closes the file

// reading the line of text from the file
OUTPUT "The file contains this line of text:"
OPEN MyFile FOR READ // opens file for reading
    READFILE, TextLine // reads a line of text from the file
    OUTPUT TextLine
CLOSEFILE(MyFile) // closes the file

```

Exam-style questions

- 1 Tick (✓) the appropriate column, in the following table, to indicate whether the statement is assignment, selection or iteration. [3]

Statement	Assignment	Selection	Iteration
IF Number1 < Number2			
Number1 ← Number2			
Number1 ← Number2 + 1			
FOR Number 1 ← 1 TO 10			
CASE OF Number1			
WHILE Number1 <> Number2			
REPEAT			

- 2 In this question you will be given a statement followed by four possible answers.

Select which of the four answers you think is correct and tick (✓) the box next to your answer.

- a) This is a Boolean operator:

- ☐ +
☐ DIV
☐ >=
☐ NOT

- b) What is meant by the term **array**?

- ☐ a list, for example 3, 2.7, TRUE, "Tulip"
☐ data structure containing elements of the same data type
☐ named data structure
☐ data structure containing elements that cannot be changed

- c) Data stored in files ...

- ☐ ... cannot be changed.
☐ ... cannot be copied.
☐ ... can be transferred to another computer.
☐ ... is volatile.

[3]

- 3 a) Explain what is meant by **selection**.

- b) Describe **two** methods of selection. Include examples of each method in your answer.

[6]

- 4 a) Explain what is meant by **iteration**.

- b) Describe **three** methods of iteration. Include examples of each method in your answer.

[8]

- 5 A sentence is to be input and stored in a variable **Sentence**.

Write pseudocode to:

- declare the variable **Sentence**
- input the sentence
- find the length of the sentence
- find out how many times the letter 'e' occurs in the sentence.

[6]

- 6 Using a programming language write a program to store the names of 50 people playing a game and their high scores. Output the names and the scores ordered from highest to lowest score and output the average score.

Your program must include and use meaningful identifier names for

- variables
- constants
- arrays
- procedures / functions.

Your program must be fully commented using your programming language's commenting feature.

[10]

9

Databases

Key objectives

The objectives of this chapter are to revise:

- the design and use of single-table databases:
 - fields, records, validation
- the use of basic data types:
 - text/alphanumeric, character, Boolean, integer, real, date/time
- the identification, purpose and use of primary keys
- SQL scripts, used to query data stored in a single-table database

9.1 Databases

REVISED

9.1.1 Single-table databases

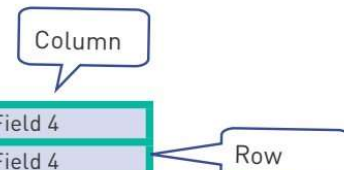
A **database** is a structured collection of data that allows people to extract information in a way that meets their needs. Data can include text, numbers and pictures. For example, a simple database of items for sale could include a description, the price and a picture of the item.

A **single-table database** contains only one table. A table consists of many **records**. The number of records in a table will vary, as new records can be added and deleted from a table as required. Each record consists of several **fields**. The number of fields in table is fixed so each record contains the same number of fields. An easy way to remember this is:

- each record is a row in the table
- each field is a column in the table.

Data is often validated when input into a field. For more on validation, see Section 7.5.

Table structure



Record 1	Field 1	Field 2	Field 3	Field 4
Record 2	Field 1	Field 2	Field 3	Field 4
Record 3	Field 1	Field 2	Field 3	Field 4
Record 4	Field 1	Field 2	Field 3	Field 4
Record 5	Field 1	Field 2	Field 3	Field 4
Record 6	Field 1	Field 2	Field 3	Field 4

9.1.2 Basic data types

A **data type** classifies how the data is stored, displayed and the operations that can be performed on the stored value. Each field requires a data type to be selected.

It is illegal to photocopy this page

These are the database data types you need to be able to use. They are available to use as *Access* data types, but the names *Access* uses may be different from the terms in the syllabus.

Data type	Description	Access data type
Text/alphanumeric	A number of characters	Short text/long text
Character	A single character	Short text with a field size of one
Boolean	One of two values: either True or False, 1 or 0, Yes or No	Yes/No
Integer	Whole number	Number formatted as fixed with zero decimal places
Real	A decimal number	Number formatted as decimal
Date/time	Date and/or time	Date/Time

9.1.3 Primary keys

Each record within a table contains data about a single item, person or event. It is important to be able to uniquely identify each record. The **primary key** is a field that uniquely identifies the record. Each primary key contains a unique value; it must contain data values that are never repeated in the table.

9.1.4 Structured Query Language

Structured Query Language (SQL) is the standard query language for writing scripts to obtain information from a database. An **SQL script** is a list of SQL commands that perform a given task.

Here are some examples of the SQL statements you need to be able to use. Here are some examples of commands.

SQL query statement	Description of statement
SELECT	Fetches specified fields (columns) from a table; queries always begin with SELECT
FROM	Identifies the table to use.
WHERE	Includes only records (rows) in a query that match a given condition.
ORDER BY	Sorts the results from a query by a given column either alphabetically or numerically.
SUM	Returns the sum of all the values in a field (column). Used with SELECT
COUNT	Counts the number of records (rows) where the field (column) matches a specified condition. Used with SELECT

Here are some examples of commands.

SELECT ItemDescription, Price FROM ItemsForSale WHERE Price > 10.00 ORDER BY Price;	Displays the description and price of all items for sale with a price of more than 10.00
SELECT SUM(Price) FROM ItemsForSale WHERE Price > 10.00	Displays the total value all items for sale with a price of more than 10.00
SELECT COUNT(Price) FROM ItemsForSale WHERE Price > 10.00	Displays the number of items for sale with a price of more than 10.00

Sample questions and answers

REVISED

A database, **StudentRoom**, was set up to show the room numbers for students living in student accommodation.

Name	Block	RoomNumber
David Smith	A	129
Su Wong	B	124
David Chow	A	129
Amy Tang	A	123
Joe Higgs	B	124
David Smith	B	125
Adel Hoo	A	125
Peter Patel	B	129

- Explain why none of the fields in the database can be used as a primary key.
- State, with a reason, a field that could be added as a primary key and give an example of some sample data for that field.
- SELECT ***
FROM StudentRoom
WHERE Block = 'A'
ORDER BY RoomNumber;
Show the output from this SQL command.

[6]

Tips

Part a) of this question involves 'explain'. This means your answer needs to give a reason why each field is not suitable to be a unique identifier. In part b), you are asked to identify an item of data that would uniquely identify the record and give the field/column a suitable name. Often if there is no suitable unique fact a short code or number is used. In part c), you are to show the output from the SQL query; make sure you show all the rows and columns required in the appropriate order that satisfy the question.

When practising questions, if you are unsure about what an SQL query would show, build the database and write the query to check your answer.

High-level sample answer

- All the fields contain duplicate values as there are many students housed in each block, some students share the same room and two of the students have the same name.
- A field that gives each student a unique code could be added, for example **StudentID**. This code should be easy to validate and have enough characters to ensure no repeats would be required, for example **ST12345**.

- | Name | Block | RoomNumber |
|-------------|-------|------------|
| Amy Tang | A | 123 |
| Adel Hoo | A | 125 |
| David Smith | A | 129 |
| David Chow | A | 129 |

It is illegal to photocopy this page

Low-level sample answer

- a) Use the Name.
- b) No need for an extra field the Name is good.
- c) Block A

Adel Hoo, 125
Amy Tang, 123
David Chow, 129
David Smith, 129

Teacher's comments

The first answer is clearly set out and readable.

For part a), all the fields have been considered and each reason is given in the context of the question. For part b), a new field with a suitable name has been identified, a reason has been given for the choice and some appropriate sample data provided. For part c), the use of a table gives the output a clear structure, only students in Block A have been shown, and they are in room number order. Where two students are sharing the room, the order is the same as in the original table.

The second answer short and some of the questions seem to have been misunderstood. For part a), the question answered is choose a suitable primary key. Not the question set but a common error. For part b), the student now sees no need for a new primary key even though the data in the name has been repeated, indicating that the completed table given was not read carefully enough. For part c), a heading and extra punctuation have been added, the block field has been removed and the ordering is on the name field.

Exam-style questions

- | | |
|--|---|
| <p>1 In this question you will be given a statement followed by four possible answers. Select which of the four answers you think is correct and tick (✓) the box next to your answer. You may need to tick more than one box.</p> <p>a) What is meant by the term table?</p> <ul style="list-style-type: none"> ☐ a collection of fields about the same item ☐ it uniquely identifies a record ☐ a single piece of data ☐ a collection of related records <p>b) What is meant by the term record?</p> <ul style="list-style-type: none"> ☐ a collection of fields about the same item ☐ a column of data ☐ a structured collection of data ☐ it uniquely identifies a row of data <p>c) What is meant by the term field?</p> <ul style="list-style-type: none"> ☐ a collection of rows about the same item ☐ it uniquely identifies a row of data ☐ a column of data ☐ a single piece of data <p style="text-align: right;">[3]</p> | <p>2 a) Explain what is meant by Structured Query Language (SQL).</p> <p>b) Describe two SQL commands.</p> <p style="text-align: right;">[6]</p> <p>3 A company that sells cakes is setting up a single-table database, to store information about cakes that can be ordered. The information includes, name of cake, number of portions, price of cake, main ingredients and decoration available.</p> <p>State, with reasons, the structure required, including names of the data items to be stored, the data type and sample data for each item.</p> <p style="text-align: right;">[8]</p> |
|--|---|

10

Boolean logic

Key objectives

The objectives of this chapter are to revise:

- the identification and use of the standard logic gates: NOT, AND, OR, NAND, NOR and XOR
- how to use logic gates to create logic circuits from:
 - a given problem
 - a logic expression
 - a truth table
- how to complete truth tables from:
 - a given problem
 - a logic expression
 - a logic circuit
- how to write a logic expression from:
 - a given problem
 - a logic circuit
 - a truth table

10.1 Standard logic gate symbols

REVISED

Electronic circuits in computers, solid-state drives and controlling devices are made up of thousands of **logic gates**. Logic gates are combined to form **logic circuits**. Logic gates take binary inputs and produce binary outputs. A **truth table** lists every binary output for every possible combination of binary inputs.

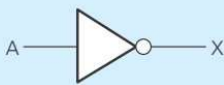
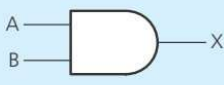
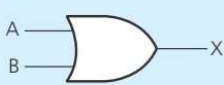
These are the six logic gates that will be considered:

NOT, AND, OR, NAND, NOR and XOR.

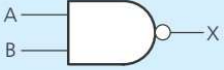
NOT is the only single input gate. All other gates are limited to two inputs for IGCSE.

10.2 The functions of the six logic gates

REVISED

Gate	Description	Truth table		Notation	
NOT 	The output, X, is 1 if: the input, A, is 0	input		X = NOT A	
		A	X	$X = \bar{A}$	
		0	1		
		1	0		
AND 	The output, X, is 1 if: both inputs, A and B, are 1	inputs		output	X = A AND B
		A	B	X	$X = A.B$
		0	0	0	
		0	1	0	
		1	0	0	
		1	1	1	
OR 	The output, X, is 1 if: either A or B, or both are 1	inputs		output	X = A OR B
		A	B	X	$X = A + B$
		0	0	0	
		0	1	1	
		1	0	1	
		1	1	1	

It is illegal to photocopy this page

Gate	Description	Truth table		Notation
NAND 	The output, X, is 1 if: input A and input B, are both not 1	inputs		$X = A \text{ NAND } B$ $X = \overline{A \cdot B}$
		A	B	
		0	0	
		0	1	
		1	0	
		1	1	
NOR 	The output, X, is 1 if: neither input A nor input B is 1	inputs		$X = A \text{ NOR } B$ $X = \overline{A + B}$
		A	B	
		0	0	
		0	1	
		1	0	
		1	1	
XOR 	The output, X, is 1 if input A is 1 and input B is 0 or The output, X, is 1 if input B is 1 and input A is 0	inputs		$X = A \text{ XOR } B$ $X = (A \cdot \overline{B}) + (\overline{A} \cdot B)$
		A	B	
		0	0	
		0	1	
		1	0	
		1	1	

Note the use of **Boolean algebra** to represent logic gates. This is optional at IGCSE but you may prefer to use this notation. In the Boolean algebra are three new symbols; these have the following meaning:

- a dot (between letters, such as **A.B**) represents the AND operation
- a plus (between letters, such as **A + B**) represents the OR operation
- a bar (above the letter or letters, such as $\overline{A}$) represents the NOT operation.

10.3 Logic circuits, logic expressions, truth tables and problem statements

REVISED

When logic gates are combined together to carry out a particular function, they form a logic circuit. You need to be able to carry out the following tasks:

- create a logic circuit from a:
 - problem statement
 - logic or Boolean expression
 - truth table
- complete a truth table from a:
 - problem statement
 - logic or Boolean expression
 - logic circuit
- write a logic or Boolean expression from a:
 - problem statement
 - logic circuit
 - truth table.

Given a problem you need to be able to represent it as a logic expression, logic circuit or truth table. You also need to be able to convert between any of the three representations. For more examples of these tasks, see *Cambridge IGCSE and O Level Computer Science Second Edition Student's Book*, pages 360–376.

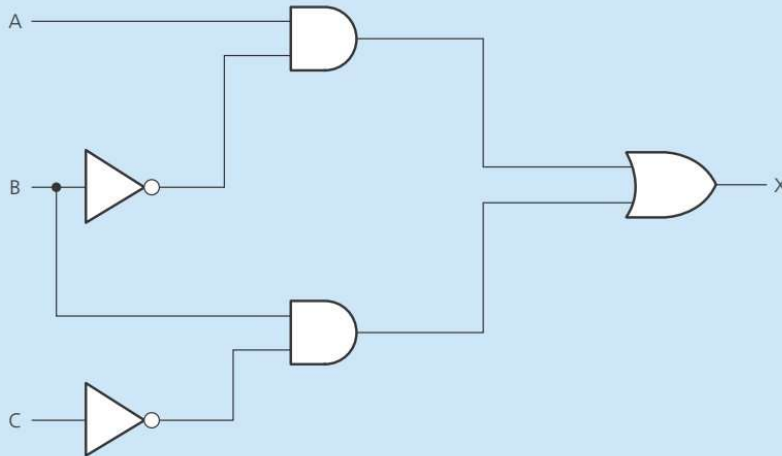
Worked example

A safety system uses three inputs to a logic circuit. An alarm, X, sounds if input A represents ON and input B represents OFF; or if input B represents ON and input C represents OFF.

Here is a logic expression that uses the inputs and output from the problem. NOT is used if the input is off.

$X = 1$ if $(A = 1 \text{ AND } B = \text{NOT } 1) \text{ OR } (B = 1 \text{ AND } C = \text{NOT } 1)$

Here is a logic circuit that uses the inputs and output from the problem. This must not be simplified.



Here is a truth table that uses the inputs and output from the problem. Every set of inputs must have an output.

Inputs			Intermediate values		Output
A	B	C	$(A = 1 \text{ AND } B = \text{NOT } 1)$	$(B = 1 \text{ AND } C = \text{NOT } 1)$	X
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	0	1	1
1	1	1	0	0	0

Sample questions and answers

REVISED

Consider this logic expression.

$$X = (A \text{ AND } B) \text{ OR } (\text{NOT } B \text{ AND } C)$$

- a) Complete the truth table for this logic expression.

Inputs			Working space		Output
A	B	C			X
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

- b) Draw a logic circuit for the un-simplified logic expression. Each logic gate must have a maximum of **two** inputs.



Tips

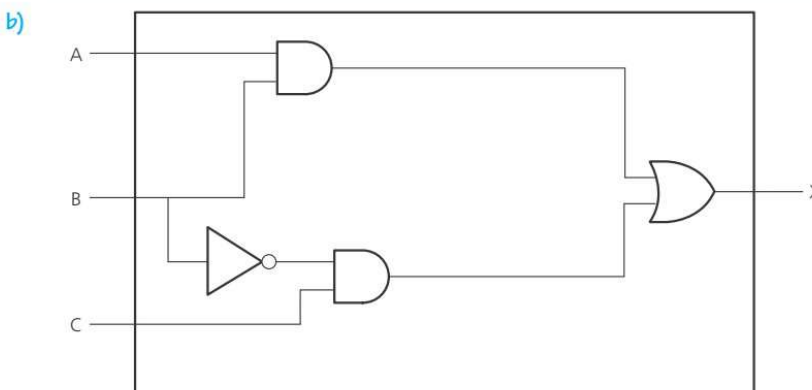
Part a) of the answer will be marked on the 0s and 1s in the last column; the middle section is for working out. It is good practice to work out the expressions in the brackets first then combine them. Take care that your 0s and 1s in the last column are clear, especially if you have made alterations to your answers. In part b), you are asked to draw a logic circuit. Make sure you use the logic gate symbols shown in the syllabus and draw the connecting lines carefully.

[8]

High-level sample answer

a)

Inputs			Working space		Output
A	B	C	A AND B	NOT B AND C	X
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	0	1



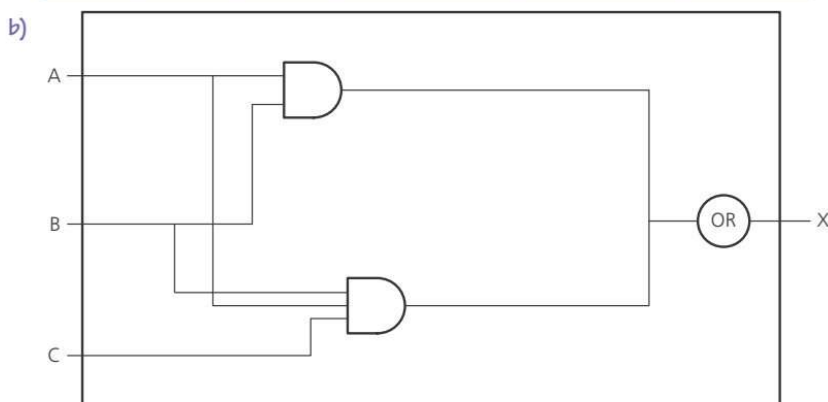
Teacher's comments

The first answer is clearly set out and readable. For part a), the intermediate stages have been included and the final column is clear correct and complete. For part b), the correct symbols have been used, the diagram is clearly set out and all the gates have two inputs or fewer.

Low-level sample answer

a)

Inputs			Working space	Output X
A	B	C		
0	0	0		0
0	0	1		
0	1	0		0
0	1	1		0
1	0	0		0
1	0	1		1
1	1	0		0
1	1	1		1



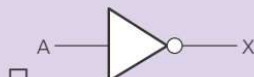
Teacher's comments

The second answer is unclear in places. For part a), the final column is incomplete, and it is unclear if there is a zero or a one on the fourth row. For part b), the lines are confusing, there is a three input AND gate, probably taken from the final row in the table, and the OR symbol is non-standard.

Exam-style questions

- 1 In this question you will be given a statement followed by four possible answers. Select which of the four answers you think is correct and tick (✓) the box next to your answer. You may need to tick more than one box.

a) What is the symbol for **AND**?



b) What is meant by the term **logic circuit**?

- ☐ a combination of logic gates with one or more outputs
☐ a single gate with on or off
☐ a table showing the output from a collection of logic gates
☐ a simplified combination of logic gates

[2]

- 2 a) i) Draw the truth table for an OR gate and an XOR gate.
 ii) State why they are different.

b) Name these gates and draw the truth tables for them.



[7]

- 3 A company offers loans to people who can answer yes to at least two of these three questions:
 – are you over 18? (input A)
 – have you over \$1000.00 in the bank? (input B)
 – have you a full-time job? (input J)

- a) Write the logic expression to represent this system.
 b) Write the truth table to represent this system.
 c) Draw the logic circuit to represent this system.

[12]